Centro Federal de Educação Tecnológica de Minas Gerais
Laboratório de Arquitetura e Organização de Computadores I
Gabriel Siqueira Silva

Diferenciador

Projeto final do processador que deriva uma função a partir de seus coeficientes gerando outro vetor

Belo Horizonte – MG
2021

- Considerações gerais

A interligação dos componentes já produzidos se tornou um trabalho de correção para que o projeto final fosse apresentado da melhor forma possível. Seguindo esse ritmo, considerou-se realizar essa última etapa da seguinte forma:

1. Interligar todos os componentes sem realizar a verificação de Clock;
2. Colocar todos os componentes em um único módulo, ou seja, o nRisc e a memória de dados estariam juntos;
3. Testar se o valor do contador do programa estava adequado e ainda, se este estava compondo todas as instruções;
4. Testar se os sinais de controle estão adequados;
5. Testar a armazenagem e as operações;
6. Realizar a lógica de retirada da memória de dados do módulo;
7. Realizar a lógica de Reset;
8. Verificar se corresponde ao propósito do projeto.

Diante dessas etapas, tentou-se ao máximo percorre-las da forma mais eficiente possível, e o que está apresentado nesse trabalho final não compõe os pontos 7 e 8. Para avaliar o sincronismo do clock, utilizou-se o sistema de Wave do ModelSim verificando cada etapa do processo e se fez necessário as seguintes alterações:

```
63      always begin @(negedge Clock)
64          begin
65              if (Reset == 1'b1)
66                  begin
67                      for (i = 0; i < 4; i = i + 1) begin
68                          RF[i] <= 8'b00000000;
69                      end
70                  end
71              else
72              begin
73                  if (RegWrite == 1'b1 & ReadMem == 1'b1)
74                  begin
75                      RF[WriteReg] <= WriteData;
76                  end
77              end
78          end
79      end
80  endmodule
```

Figura 1 - Nova etapa da escrita em registradores

```
43      always @(posedge Clock) begin
44      /*
45          if (Reset == 1'b1) begin
46              MemReadData <= 0;
47              for (i = 0; i < 256; i = i + 1) begin
48                  RAM[i] <= 8'b00000000;
49              end
50          end
51          else
52          begin
53          */
54              if (MemWrite == 1'b1) begin
55                  RAM[Address] <= MemWriteData;
56              end
57              if (MemRead == 1'b1) begin
58                  MemReadData <= RAM[Address];
59              end
60
61
62          //end
63      end
64      /*
```

Figura 2 - Alteração no código da memória

Na análise das ondas, verificou que algumas leituras estavam ocorrendo de forma normal, mas a sua armazenagem em um registrador não estava ocorrendo da maneira correta. Dessa forma, colocou-se a escrita e leitura em memória em um mesmo clock, enquanto que a escrita em registradores, após uma leitura sempre vai acontecer em uma borda de descida do clock diferenciando da escrita padrão do banco de registradores proposto nos relatórios anteriores. Isso é possível pois, ainda na avaliação de ondas, MemWrite e MemRead nunca serão 1 ao mesmo tempo, dessa forma, não há priorização na escrita ou na leitura. Então, nessa atualização, a memória de dados funcionará sempre em borda de subida e o banco de registradores funcionará nas duas bordas.

- Um problema com saltos

Como o código utiliza muitos saltos seguidos, é inevitável que ocorresse um salto e o seu subsequente ocorresse sem necessidade, para superar esse obstáculo, as instruções já armazenadas se alteraram em sua quantidade:

```
42          rom[10] = 8'b10100111;
43          rom[11] = 8'b10101001;
44          rom[12] = 8'b11000100;
45          rom[13] = 8'b00000000;
46          rom[14] = 8'b11001001;
47          rom[15] = 8'b11001110;
48          rom[16] = 8'b00000000;
49          rom[17] = 8'b11010101;
50          rom[18] = 8'b01100111;
51          rom[19] = 8'b10000101;
52          rom[20] = 8'b00111000;
53          rom[21] = 8'b01101001;
54          rom[22] = 8'b10000110;
55          rom[23] = 8'b00111001;
56          rom[24] = 8'b01100110;
```

Figura 3 - Adição de 0's entre instruções

O salto incondicional é representado pelo OPCode 110, observe que na linha 48 a uma entrada totalmente zerada. Isso foi necessário pois o jump representado na linha 49 é um jump de retorno (Utiliza números negativos), enquanto na linha 47 é um jump de avanço (Apenas números positivos), e quando estavam próximos causava um *loop* infinito no sistema, fazendo com que ele não termine. A adição de instruções zeradas foi necessária em muitas partes do código dessa forma suprindo os possíveis *loops* infinitos. Esse processo auxiliou na verificação na passagem de PC, analisando se o mesmo passa por todas as instruções.

- nRISC e memória

Conforme estabelecido no tópico anterior, o processador e a memória estavam unidos inicialmente, mas posteriormente sofreram uma separação e a situação está descrita abaixo:

```
module Datapath(Clk, Reset, ALUResult, Data2, MemWrite, MemRead, RAMResult);

input Clk, Reset;
wire [7:0] PCNext, PCBeq;

output [7:0] ALUResult, Data2, RAMResult;
output MemWrite, MemRead;

wire [7:0] Inst;
// Sinais
wire PCWrite, Regdst, Jump, Branch, MemRead, MemWrite, MemtoReg, RegWrite;
wire [1:0] ALUOp, ALUSrc;

// Do registrador
wire [1:0] MUXReg;
wire [7:0] Data1, Data2;

//ALU
wire [7:0] Signal2to8, Signal5to8, MUXAlu;
wire [7:0] ALUResult;
wire SignalZero, MUXBranch;

// PC
reg [7:0] PC;
wire [7:0] MUX_ALU_MEMtoReg, RAMResult;

// PC
initial begin
    PC <= 8'b00000000;
end
```

```
// Memoria de instrucao
ROM bloco1(.Clock(Clk), .Instruction(Inst), .Address(PC));

// Controle
Ctrl bloco2(Inst[7:5], Reset, Clk, PCWrite, Regdst, Jump, Branch, MemRead, MemWrite, MemtoReg, RegWrite, ALUOp, ALUSrc);

// MUX proximo ao registrador
Mux2to1Reg bloco3(Inst[3:2], Inst[1:0], Regdst, MUXReg);

// Register File
registerFile bloco4(Inst[4], Inst[3:2], MUXReg, MUX_ALU_MEMtoReg, RegWrite, Data1, Data2, Clk, Reset, MemRead);

// Intermediaries
SignExtension2bits bloco5(Inst[1:0], Signal2to8);
SignExtension5bits bloco6(Inst[4:0], Signal5to8);
Mux3to1 bloco7(Data2, Signal2to8, Inst[1:0], ALUSrc, MUXAlu);

// ALU
ALU bloco8(Data1, MUXAlu, ALUOp, ALUResult, SignalZero);

Mux2to1 bloco10(ALUResult, RAMResult, MemtoReg, MUX_ALU_MEMtoReg);

assign MUXBranch = SignalZero & Branch;
```

```
always @(posedge Clk) begin
    if(Inst[7:5] != 3'b111)begin
        case({MUXBranch,Jump})
            2'b00:  PC <= PC + 8'b00000001;
            2'b01:  PC <= PC + 8'b00000001 + Signal5to8;
            2'b10:  PC <= PC + 8'b00000001 + Signal2to8;
            default: ;
        endcase
    end
end

endmodule
```

Figura 4 - nRISC

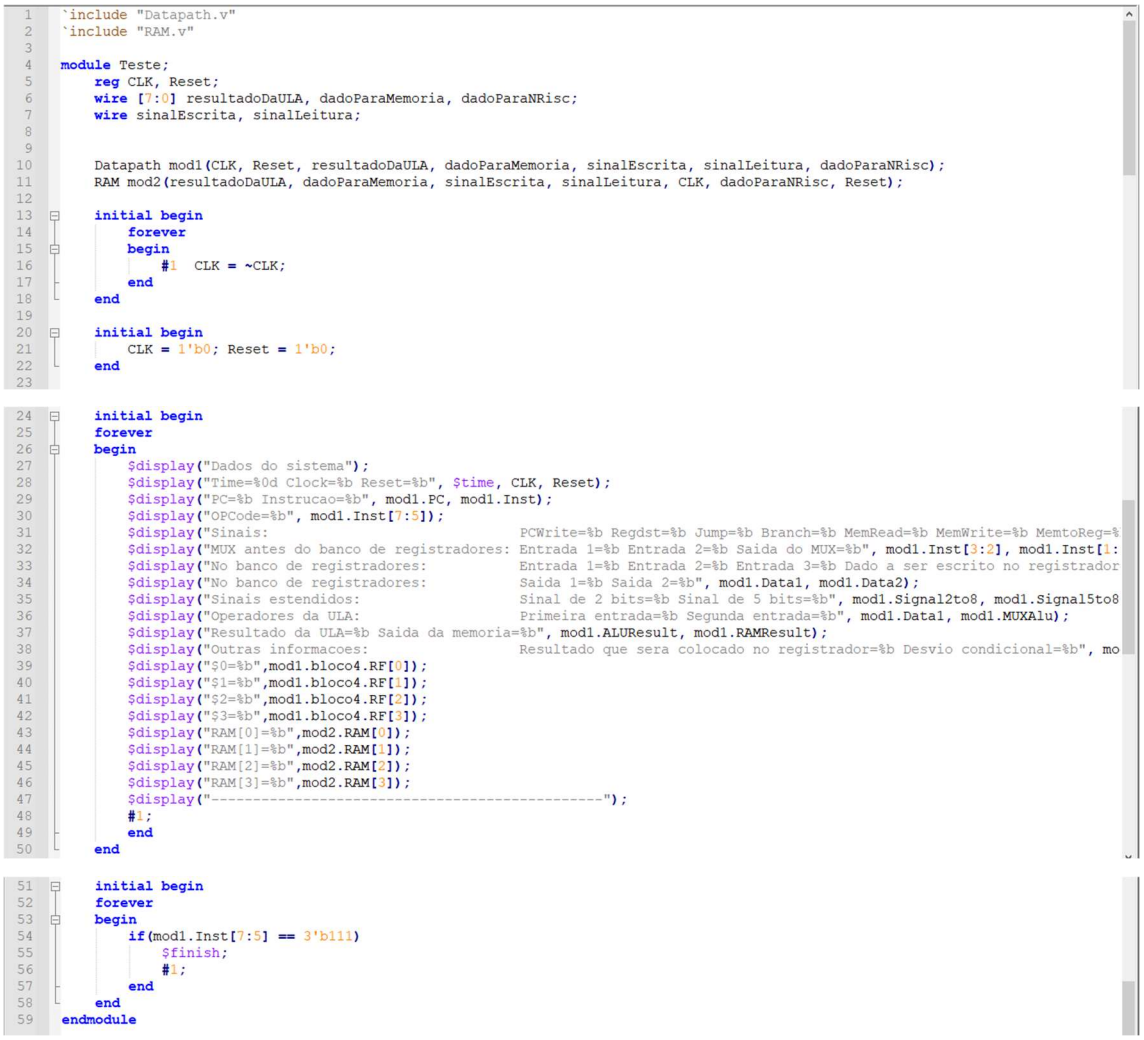
```
module RAM (Address, MemWriteData, MemWrite, MemRead, Clock, MemReadData, Reset);

    input Reset;
    input wire [7:0] Address;
    input wire [7:0] MemWriteData;
    input wire MemWrite, MemRead;
    input wire Clock;
    output reg [7:0] MemReadData;

    reg [7:0] RAM[255:0];
    integer i;
/*
    initial begin
        MemReadData <= 0;
        for (i = 0; i < 256; i = i + 1) begin
            RAM[i] <= 8'b00000000;
        end
    end
*/
    always @(posedge Clock) begin
    /*
        if (Reset == 1'b1) begin
            MemReadData <= 0;
            for (i = 0; i < 256; i = i + 1) begin
                RAM[i] <= 8'b00000000;
            end
        end
        else
        begin
        */
            if (MemWrite == 1'b1) begin
                RAM[Address] <= MemWriteData;
            end
            if (MemRead == 1'b1) begin
                MemReadData <= RAM[Address];
            end


        //end
    end
    /*
    always @(negedge Clock) begin
        if (Reset == 1'b1) begin
            MemReadData <= 0;
            for (i = 0; i < 256; i = i + 1) begin
                RAM[i] <= 8'b00000000;
            end
        end
        else
        begin
            if (MemRead == 1'b1) begin
                MemReadData <= RAM[Address];
            end
        end
    end
    */
endmodule
```

Figura 5 - Memória RAM

O módulo Datapath compõe tudo que foi escrito para este trabalho final, dessa forma, foi necessário declarar vários fios para que ocorresse a interligação dos módulos. É importante destacar que nesse Datapath, alguns módulos foram resumidos de forma que os componentes de soma e And se apresentam apenas com os operadores “+” e “&” juntamente com a lógica predefinida.

- ModelSim

A simulação no ModelSim será do módulo de Teste que está abaixo e análise se dará nos melhores tempos:



```
`include "Datapath.v"
`include "RAM.v"

module Teste;
    reg CLK, Reset;
    wire [7:0] resultadoDaULA, dadoParaMemoria, dadoParaNRisc;
    wire sinalEscrita, sinalLeitura;


    Datapath mod1(CLK, Reset, resultadoDaULA, dadoParaMemoria, sinalEscrita, sinalLeitura, dadoParaNRisc);
    RAM mod2(resultadoDaULA, dadoParaMemoria, sinalEscrita, sinalLeitura, CLK, dadoParaNRisc, Reset);

    initial begin
        forever
        begin
            #1  CLK = ~CLK;
        end
    end

    initial begin
        CLK = 1'b0; Reset = 1'b0;
    end

    initial begin
    forever
    begin
        $display("Dados do sistema");
        $display("Time=%0d Clock=%b Reset=%b", $time, CLK, Reset);
        $display("PC=%b Instrucao=%b", mod1.PC, mod1.Inst);
        $display("OPCode=%b", mod1.Inst[7:5]);
        $display("Sinais:                              PCWrite=%b Regdst=%b Jump=%b Branch=%b MemRead=%b MemWrite=%b MemtoReg=%
        $display("MUX antes do banco de registradores: Entrada 1=%b Entrada 2=%b Saida do MUX=%b", mod1.Inst[3:2], mod1.Inst[1:
        $display("No banco de registradores:           Entrada 1=%b Entrada 2=%b Entrada 3=%b Dado a ser escrito no registrador
        $display("No banco de registradores:           Saida 1=%b Saida 2=%b", mod1.Data1, mod1.Data2);
        $display("Sinais estendidos:                   Sinal de 2 bits=%b Sinal de 5 bits=%b", mod1.Signal2to8, mod1.Signal5to8
        $display("Operadores da ULA:                   Primeira entrada=%b Segunda entrada=%b", mod1.Data1, mod1.MUXAlu);
        $display("Resultado da ULA=%b Saida da memoria=%b", mod1.ALUResult, mod1.RAMResult);
        $display("Outras informacoes:                  Resultado que sera colocado no registrador=%b Desvio condicional=%b", mo
        $display("$0=%b",mod1.bloco4.RF[0]);
        $display("$1=%b",mod1.bloco4.RF[1]);
        $display("$2=%b",mod1.bloco4.RF[2]);
        $display("$3=%b",mod1.bloco4.RF[3]);
        $display("RAM[0]=%b",mod2.RAM[0]);
        $display("RAM[1]=%b",mod2.RAM[1]);
        $display("RAM[2]=%b",mod2.RAM[2]);
        $display("RAM[3]=%b",mod2.RAM[3]);
        $display("-----------------------------------------------");
        #1;
        end
    end

    initial begin
    forever
    begin
        if(mod1.Inst[7:5] == 3'b111)
            $finish;
            #1;
        end
    end
endmodule
```

Figura 6 - Módulo teste

No módulo teste é importante notar como está ocorrendo o processo de Clock, onde este começa em 0 e a cada 1ps (picossegundo) altera o seu valor de 0 para 1 e vice-versa. Além disso, no módulo de teste ocorre a parada do processo com a função $finish e vai acontecer quando o OPCode da instrução for 111. Os outros campos do módulo são os displays que irão mostrar a cada tempo o que está acontecendo em cada local do sistema.

- Resultados da simulação do módulo Teste

Para essa simulação foram compilados todos os módulos já realizados:

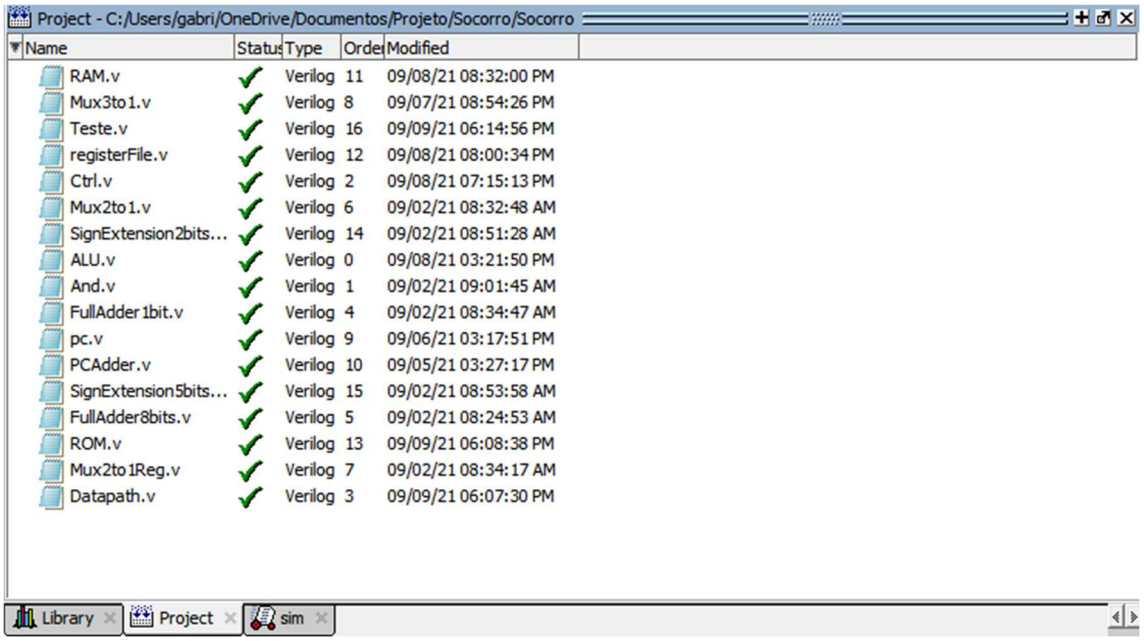


Figura 7 - Todos os componentes

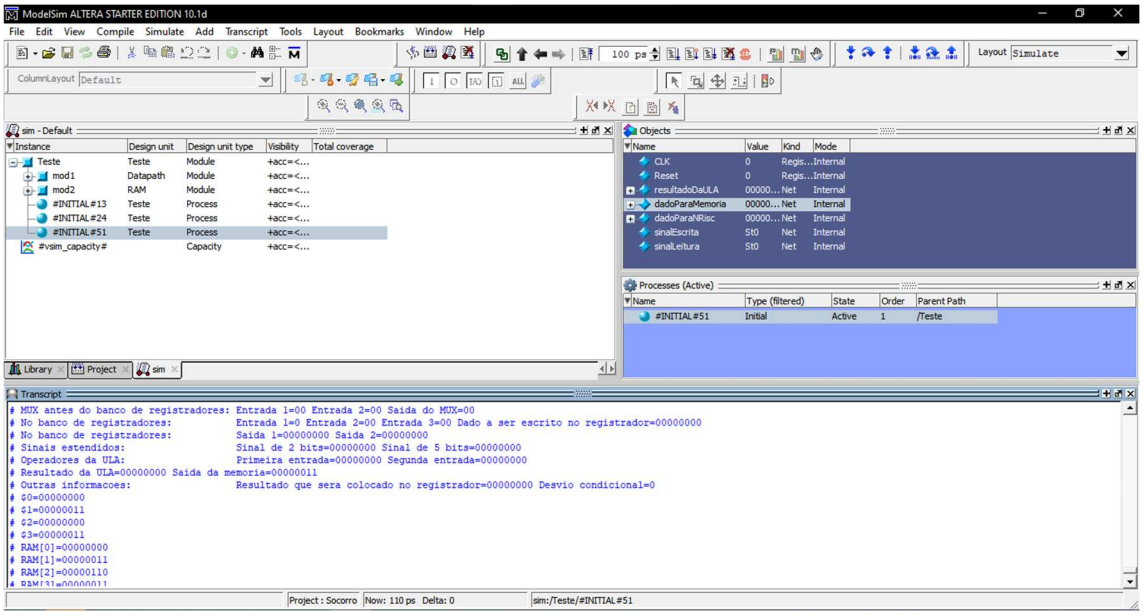


Figura 8 - Layout completo

Dos resultados determinados pelo display, é possível separar os tempos mais importantes, como Time = 20, onde na RAM, já estão os coeficientes da equação:

$3x^2 + 3x + 2 \rightarrow RAM[2] = 3, RAM[1] = 3, RAM[0] = 2$, e ainda nesse mesmo tempo, o tamanho já foi definido e armazenado no registrador $3.

```
# Time=20 Clock=0 Reset=0
# PC=00001010 Instrucao=01011110
# OPCode=010
# Sinais:                              PCWrite=1 Regdst=0 Jump=0 Branch=0 MemRead=0 MemWrite=0 MemtoReg=0 RegWrite=1 ALUOp=00 ALUSrc=01
# MUX antes do banco de registradores: Entrada 1=11 Entrada 2=10 Saida do MUX=11
# No banco de registradores:           Entrada 1=1 Entrada 2=11 Entrada 3=11 Dado a ser escrito no registrador=00000010
# No banco de registradores:           Saida 1=00000000 Saida 2=00000011
# Sinais estendidos:                   Sinal de 2 bits=00000010 Sinal de 5 bits=11111110
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000010
# Resultado da ULA=00000010 Saida da memoria=xxxxxxxx
# Outras informacoes:                  Resultado que sera colocado no registrador=00000010 Desvio condicional=0
# $0=00000000
# $1=00000000
# $2=00000011
# $3=00000011
# RAM[0]=00000010
# RAM[1]=00000011
# RAM[2]=00000011
# RAM[3]=xxxxxxxx
```

Figura 9 - Preparo para as derivadas

Em Time = 36, ocorre o primeiro processo de derivação:

```
# Time=36 Clock=0 Reset=0
# PC=00100011 Instrucao=00010101
# OPCode=000
# Sinais:                              PCWrite=1 Regdst=0 Jump=0 Branch=0 MemRead=0 MemWrite=1 MemtoReg=0 RegWrite=0 ALUOp=00 ALUSrc=01
# MUX antes do banco de registradores: Entrada 1=01 Entrada 2=01 Saida do MUX=01
# No banco de registradores:           Entrada 1=1 Entrada 2=01 Entrada 3=01 Dado a ser escrito no registrador=00000001
# No banco de registradores:           Saida 1=00000000 Saida 2=00000000
# Sinais estendidos:                   Sinal de 2 bits=00000001 Sinal de 5 bits=11110101
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000001
# Resultado da ULA=00000001 Saida da memoria=00000010
# Outras informacoes:                  Resultado que sera colocado no registrador=00000001 Desvio condicional=0
# $0=00000000
# $1=00000000
# $2=00000000
# $3=00000011
# RAM[0]=00000000
# RAM[1]=00000011
# RAM[2]=00000011
# RAM[3]=xxxxxxxx
# ------------------------------------------------
```

Figura 10 – Primeira posição do vetor RAM, ocorreu a derivada

No tempo de 68ps se tem o primeiro deslocamento de bits, representando a derivada do segundo termo:

```
# Time=68 Clock=0 Reset=0
# PC=00010101 Instrucao=00111000
# OPCode=001
# Sinais:                              PCWrite=1 Regdst=0 Jump=0 Branch=0 MemRead=1 MemWrite=0 MemtoReg=1 RegWrite=1 ALUOp=00 ALUSrc=01
# MUX antes do banco de registradores: Entrada 1=10 Entrada 2=00 Saida do MUX=10
# No banco de registradores:           Entrada 1=1 Entrada 2=10 Entrada 3=10 Dado a ser escrito no registrador=00000011
# No banco de registradores:           Saida 1=00000000 Saida 2=00000001
# Sinais estendidos:                   Sinal de 2 bits=00000000 Sinal de 5 bits=11111000
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000000
# Resultado da ULA=00000000 Saida da memoria=00000011
# Outras informacoes:                  Resultado que sera colocado no registrador=00000011 Desvio condicional=0
# $0=00000000
# $1=00000000
# $2=00000001
# $3=00000011
# RAM[0]=00000000
# RAM[1]=00000011
# RAM[2]=00000011
# RAM[3]=00000001
```

Figura 11 - Deslocamento de bits

Por fim entre [75-78]ps acontece a última derivação, gerando o resultado final no tempo de 78ps fazendo com que o programa precise apenas terminar:

```
# Time=78 Clock=0 Reset=0
# PC=00011010 Instrucao=10000111
# OPCode=100
# Sinais:                              PCWrite=1 Regdst=0 Jump=0 Branch=0 MemRead=0 MemWrite=1 MemtoReg=0 RegWrite=0 ALUOp=00 ALUSrc=01
# MUX antes do banco de registradores: Entrada 1=01 Entrada 2=11 Saida do MUX=01
# No banco de registradores:           Entrada 1=0 Entrada 2=01 Entrada 3=01 Dado a ser escrito no registrador=00000011
# No banco de registradores:           Saida 1=00000000 Saida 2=00000110
# Sinais estendidos:                   Sinal de 2 bits=00000011 Sinal de 5 bits=00000111
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000011
# Resultado da ULA=00000011 Saida da memoria=00000011
# Outras informacoes:                  Resultado que sera colocado no registrador=00000011 Desvio condicional=0
# $0=00000000
# $1=00000110
# $2=00000011
# $3=00000011
# RAM[0]=00000000
# RAM[1]=00000011
# RAM[2]=00000110
# RAM[3]=00000001
*
```

Figura 12 - Derivada completa

Essa finalização do processo ocorre no tempo de 98ps, onde acontece o último desvio condicional do sistema:

```
# Time=98 Clock=0 Reset=0
# PC=00001100 Instrucao=10101001
# OPCode=101
# Sinais:                              PCWrite=1 Regdst=0 Jump=0 Branch=1 MemRead=0 MemWrite=0 MemtoReg=0 RegWrite=0 ALUOp=01 ALUSrc=00
# MUX antes do banco de registradores: Entrada 1=10 Entrada 2=01 Saida do MUX=10
# No banco de registradores:           Entrada 1=0 Entrada 2=10 Entrada 3=10 Dado a ser escrito no registrador=00000000
# No banco de registradores:           Saida 1=00000000 Saida 2=00000000
# Sinais estendidos:                   Sinal de 2 bits=00000001 Sinal de 5 bits=00001001
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000000
# Resultado da ULA=00000000 Saida da memoria=00000001
# Outras informacoes:                  Resultado que sera colocado no registrador=00000000 Desvio condicional=1
# $0=00000000
# $1=00000011
# $2=00000000
# $3=00000011
# RAM[0]=00000000
# RAM[1]=00000011
# RAM[2]=00000110
# RAM[3]=00000011
# ------------------------------------------------
```

Figura 13 - Desvio condicional

E a partir de uma sequência de saltos nos próximos tempos, o programa é finalizado com 110ps.

```
# Time=110 Clock=0 Reset=0
# PC=00010110 Instrucao=11100000
# OPCode=111
# Sinais:                              PCWrite=1 Regdst=0 Jump=1 Branch=0 MemRead=0 MemWrite=0 MemtoReg=0 RegWrite=0 ALUOp=00 ALUSrc=00
# MUX antes do banco de registradores: Entrada 1=00 Entrada 2=00 Saida do MUX=00
# No banco de registradores:           Entrada 1=0 Entrada 2=00 Entrada 3=00 Dado a ser escrito no registrador=00000000
# No banco de registradores:           Saida 1=00000000 Saida 2=00000000
# Sinais estendidos:                   Sinal de 2 bits=00000000 Sinal de 5 bits=00000000
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000000
# Resultado da ULA=00000000 Saida da memoria=00000011
# Outras informacoes:                  Resultado que sera colocado no registrador=00000000 Desvio condicional=0
# $0=00000000
# $1=00000011
# $2=00000000
# $3=00000011
# RAM[0]=00000000
# RAM[1]=00000011
# RAM[2]=00000110
# RAM[3]=00000011
*
```

Figura 14 - Halt

$$Entrada: [3,3,2] \rightarrow Saída: [6,3,0]$$

Mesmo com a saída satisfatória, o processo não foi o adequado. Isso acontece pois foi observado que as entradas válidas são aquelas que representam o seguinte tipo de equação:

$$ax^2 + ax + c,$$

diferente do que foi proposto inicialmente. Veja o que acontece se a entrada for [2,2,2].

```
# Time=110 Clock=0 Reset=0
# PC=00010110 Instrucao=11100000
# OPCode=111
# Sinais:                              PCWrite=1 Regdst=0 Jump=1 Branch=0 MemRead=0 MemWrite=0 MemtoReg=0 RegWrite=0 ALUOp=00 ALUSrc=00
# MUX antes do banco de registradores: Entrada 1=00 Entrada 2=00 Saida do MUX=00
# No banco de registradores:           Entrada 1=0 Entrada 2=00 Entrada 3=00 Dado a ser escrito no registrador=00000000
# No banco de registradores:           Saida 1=00000000 Saida 2=00000000
# Sinais estendidos:                   Sinal de 2 bits=00000000 Sinal de 5 bits=00000000
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000000
# Resultado da ULA=00000000 Saida da memoria=00000010
# Outras informacoes:                  Resultado que sera colocado no registrador=00000000 Desvio condicional=0
# $0=00000000
# $1=00000010
# $2=00000000
# $3=00000011
# RAM[0]=00000000
# RAM[1]=00000010
# RAM[2]=00000100
# RAM[3]=00000011
# ------------------------------------------------
```

Figura 15 - Resultado da entrada [2,2,2]

$$Entrada\text{: } [2,2,2] \rightarrow Saída\text{: } [4,2,0]$$

Quando a entrada for [3,2,2], a saída vai ser a mesma que a entrada [2,2,2].

```
# Time=110 Clock=0 Reset=0
# PC=00010110 Instrucao=11100000
# OPCode=111
# Sinais:                              PCWrite=1 Regdst=0 Jump=1 Branch=0 MemRead=0 MemWrite=0 MemtoReg=0 RegWrite=0 ALUOp=00 ALUSrc=00
# MUX antes do banco de registradores: Entrada 1=00 Entrada 2=00 Saida do MUX=00
# No banco de registradores:           Entrada 1=0 Entrada 2=00 Entrada 3=00 Dado a ser escrito no registrador=00000000
# No banco de registradores:           Saida 1=00000000 Saida 2=00000000
# Sinais estendidos:                   Sinal de 2 bits=00000000 Sinal de 5 bits=00000000
# Operadores da ULA:                   Primeira entrada=00000000 Segunda entrada=00000000
# Resultado da ULA=00000000 Saida da memoria=00000010
# Outras informacoes:                  Resultado que sera colocado no registrador=00000000 Desvio condicional=0
# $0=00000000
# $1=00000010
# $2=00000000
# $3=00000011
# RAM[0]=00000000
# RAM[1]=00000010
# RAM[2]=00000100
# RAM[3]=00000011
# ------------------------------------------------
```

Figura 16 - Resultado da entrada [3,2,2]

A origem deste problema está interligada com a escrita nos registradores e a utilização do deslocamento de bits. Ocorre uma demora para acessar todas as funcionalidades.